API Gateway Configuration Guide

Version 10.3

October 2018

WEBMETHODS

This document applies to webMethods API Gateway 10.3 and to all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Document ID: YAI-CG-103-20230128

Table of Contents

About this Guide

This guide describes how you can install, and configure API Gateway and other API Gateway components to effectively manage APIs for services that you want to expose to consumers, whether inside your organization or outside to partners and third parties.

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
Narrowfont	Identifies service names and locations in the format *folder.subfolder.service*, APIs, Java classes, methods, properties.
Italic	Identifies: Variables for which you must supply values specific to your own situation or environment. New terms the first time they occur in the text. References to other documentation sources.
`Monospace font`	Identifies: Text you must type in. Messages displayed by the system. Program code.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
|	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the | symbol.
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information and Support

Software AG Documentation Website

You can find documentation on the Software AG Documentation website at http://documentation.softwareag.com. The site requires credentials for Software AG's Product Support site Empower. If you do not have Empower credentials, you must use the TECHcommunity website.

Software AG Empower Product Support Website

If you do not yet have an account for Empower, send an email to empower@softwareag.com with your name, company, and company email address and request an account.

Once you have an account, you can open Support Incidents online via the eService section of Empower at https://empower.softwareag.com/.

You can find product information on the Software AG Empower Product Support website at https://empower.softwareag.com.

To submit feature/enhancement requests, get information about product availability, and download products, go to Products.

To get information about fixes and to read early warnings, technical papers, and knowledge base articles, go to the Knowledge Center.

If you have any questions, you can find a local or toll-free number for your country in our Global Support Contact Directory at https://empower.softwareag.com/public_directory.asp and give us a call.

Software AG TECHcommunity

You can find documentation and other technical information on the Software AG TECHcommunity website at http://techcommunity.softwareag.com. You can:

- Access product documentation, if you have TECHcommunity credentials. If you do not, you will need to register and specify "Documentation" as an area of interest.
- Access articles, code samples, demos, and tutorials.
- Use the online discussion forums, moderated by Software AG professionals, to ask questions, discuss best practices, and learn how other customers are using Software AG technology.
- Link to external websites that discuss open standards and web technology.

Data Protection

Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

1 API Gateway Architecture

API Gateway Editions

API Gateway is available in two editions based on the type of license used:

- **API Gateway: Standard Edition**. This edition of API Gateway offers only API protection.
- **API Gateway: Advanced Edition**. This edition of API Gateway offers both API protection and mediation capabilities.

You can view the type of license by selecting ***Username*** **> About**. The information is displayed under Product Information section. You can change the type of license at any time from the Standard Edition to the Advanced Edition.

Note:
For details on API Gateway License management see, *webMethods Integration Server Administrator's Guide*.

This table lists the capabilities available in the Standard and the Advanced Editions of API Gateway.

Feature	Standard Edition	Advanced Edition
Users and Roles	Administrators	Administrators and API Provider
Administration ■ Ports ■ License management ■ Load balancing ■ Keystore configuration	Yes	Yes
Administration ■ Extended settings	No	Yes
Alias management	No	Yes
Service management	No	Yes
Policy management ■ Threat protection rules	Yes	Yes
Policy management ■ Global policies ■ Policy templates	No	Yes
Export and Import	No	Yes

Feature	Standard Edition	Advanced Edition
■ APIs ■ Global policies		
Application management	No	Yes
Plans and packages	No	Yes
Analytics ■ Threat protection rule violations	Yes	Yes
Analytics ■ Service ■ Applications ■ Consumers	No	Yes
Clustering and auto synchronization	No	Yes

API Gateway Deployment Scenarios

API Gateway enforces threat protection, policies and routing capabilities for APIs. This section describes high-level API Gateway architecture for various deployment scenarios.

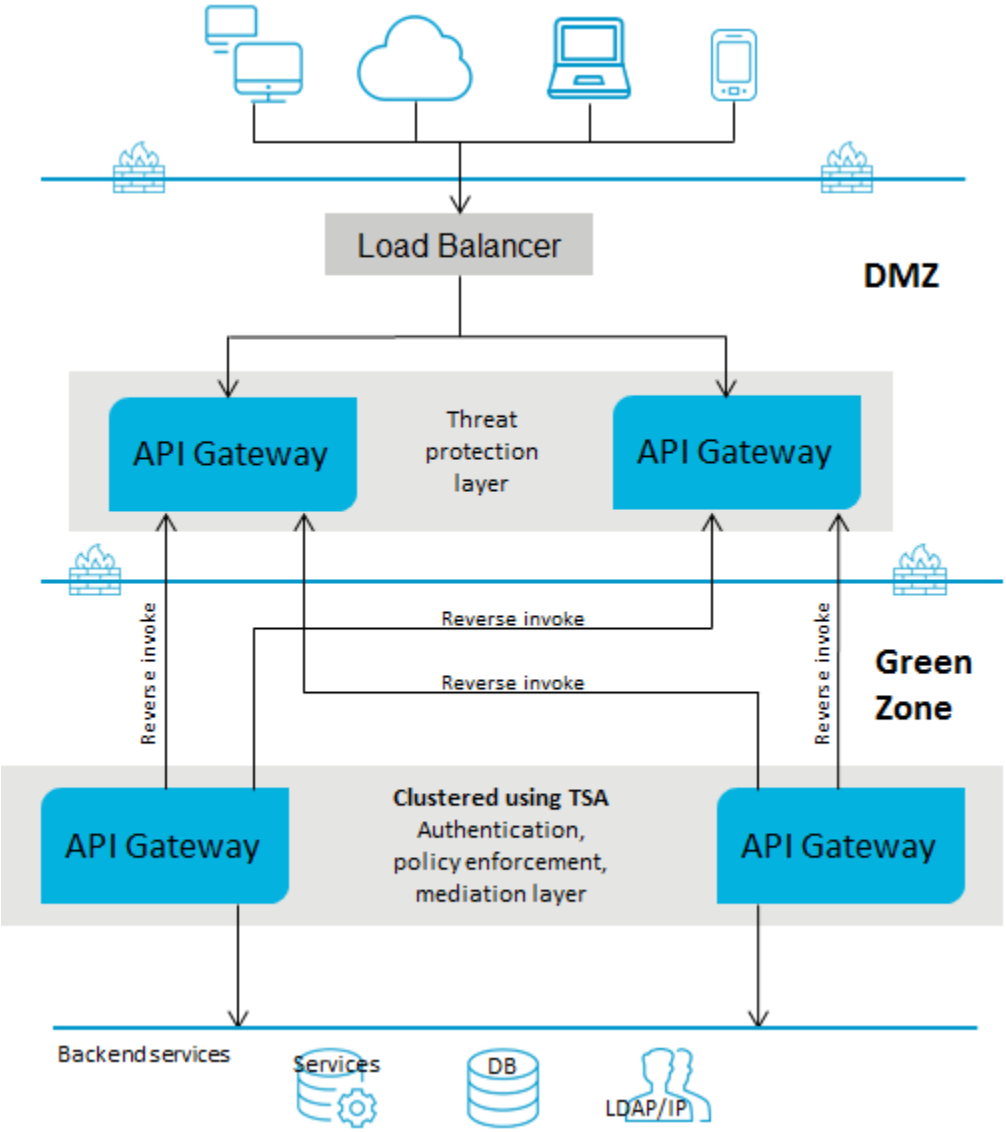
Deployment scenario 1: Paired gateway deployment

This setup consists of:

- One or more standard edition API Gateways for threat protection and connected to a load balancer in DMZ.
- One or more advanced version API Gateways clustered in the green zone to enforce policies and provide routing capabilities. You can have multiple instances of API Gateways connected through a load balancer and clustered using Terracotta Server Array. You can add an extra layer of protection by using reverse invoke.

A firewall protects the API Gateway infrastructure in the paired deployment. You can add an extra layer of protection by using reverse invoke. The API Gateways communicate between the zones using the reverse invoke approach.

The following diagram provides an architectural overview of the paired gateway deployment:



Note:
If you have multiple instances of API Gateway connected using a load balancer for threat protection and you change the enforced rules on one of the API Gateway instances, you must restart the other instances to synchronize the rule enforcement across all the API Gateway instances.

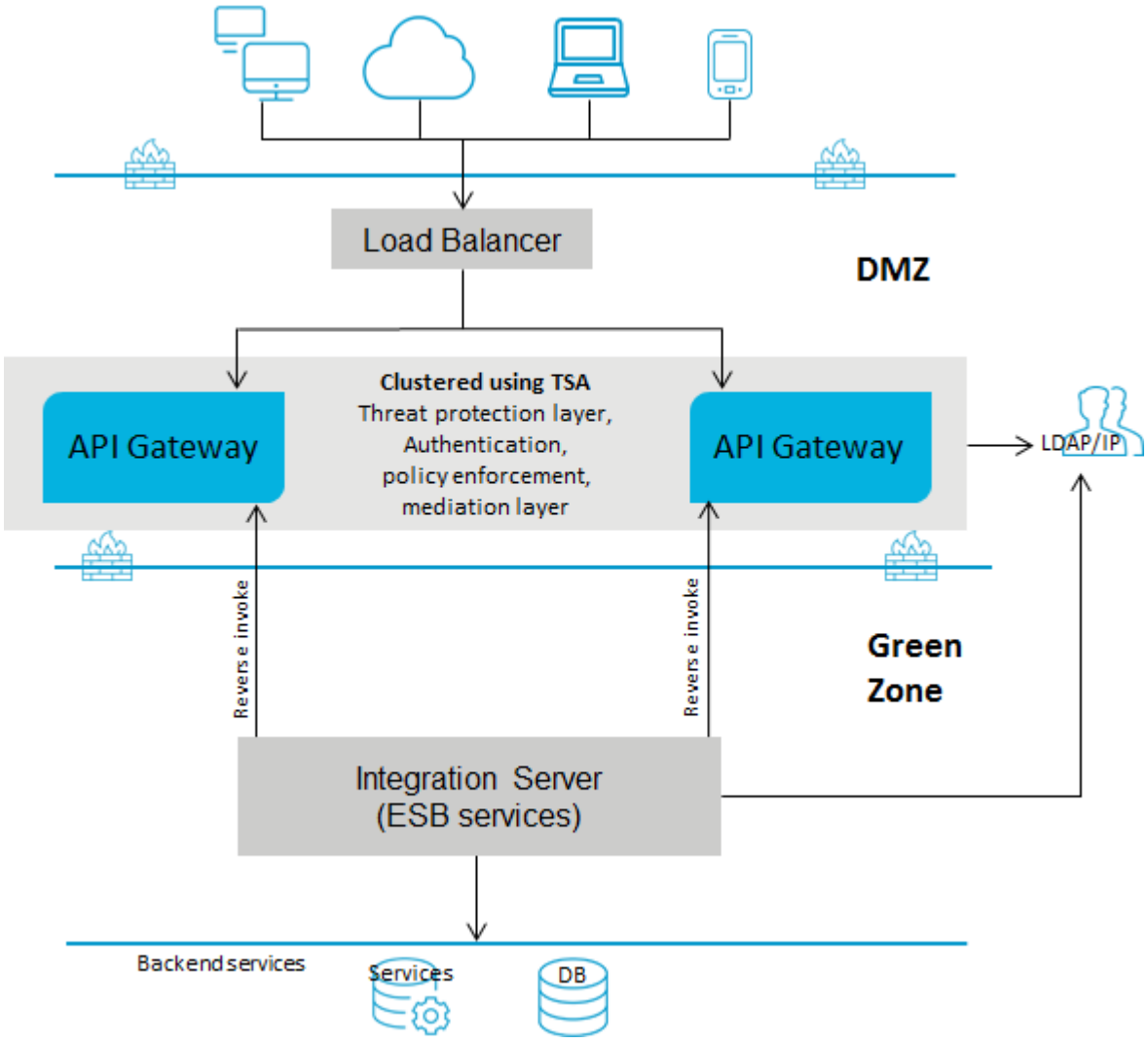
Deployment scenario 2: Single gateway in the DMZ for webMethods customers

This setup consists of:

- One or more advanced edition API Gateways clustered and connected to a load balancer in DMZ. You can have multiple instances of API Gateways connected through a load balancer and clustered using Terracotta Server Array. A single API Gateway is used for enforcing authentication and routing capabilities.
- The ESB services in Integration Server reside in the green zone behind the firewall.

If you use reverse invoke for communication between API Gateway and the internal ESB, ensure that the endpoint in the routing policy applied is configured as apigateway://*registrationPort-aliasname/relative path* of the service. For details, see the Ports section and the Routing policies section in the *webMethods API Gateway User's Guide*.

The following diagram provides an architectural overview of the API Gateway deployment in a DMZ for webMethods customers:



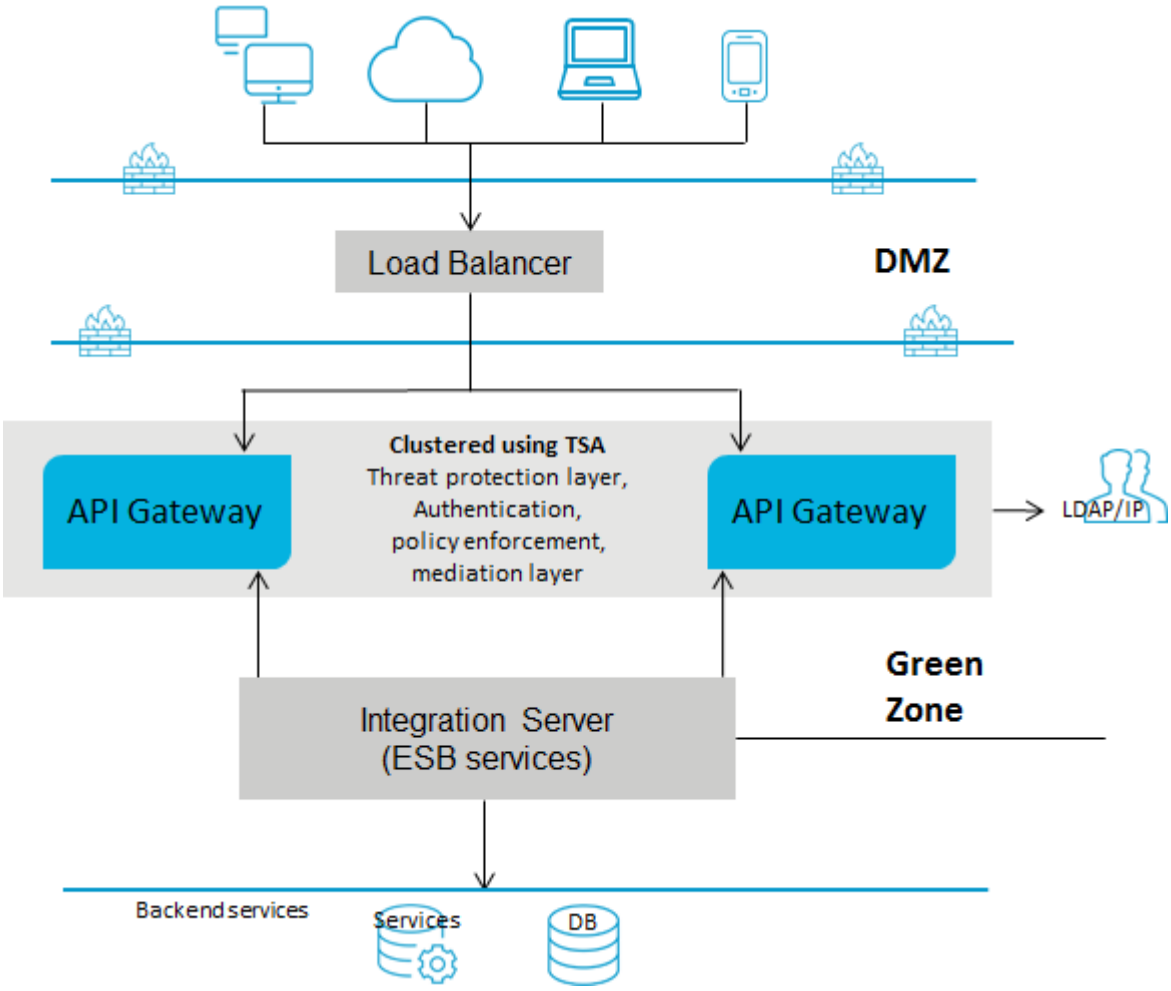
Deployment scenario 3: Single gateway in the green zone for webMethods customers

This setup consists of:

- One or more advanced edition API Gateways clustered in the green zone and connected to a load balancer in DMZ. A single API Gateway is used for enforcing authentication and routing capabilities. This deployment does not require threat protection. However, you can configure and enforce threat protection, if required. You can have multiple instances of API Gateways connected through a load balancer and clustered using Terracotta Server Array.

- The ESB services in Integration Server reside in the green zone behind the firewall. Because the API Gateway and the ESB services reside in the green zone, the ESB services are directly invoked.

The following diagram provides an architectural overview of the API Gateway deployment in the green zone for webMethods customers:



Note:
Because the API Gateway instance and the ESB service are in the same network, you can either directly invoke the ESB service or use the reverse invoke approach as required.

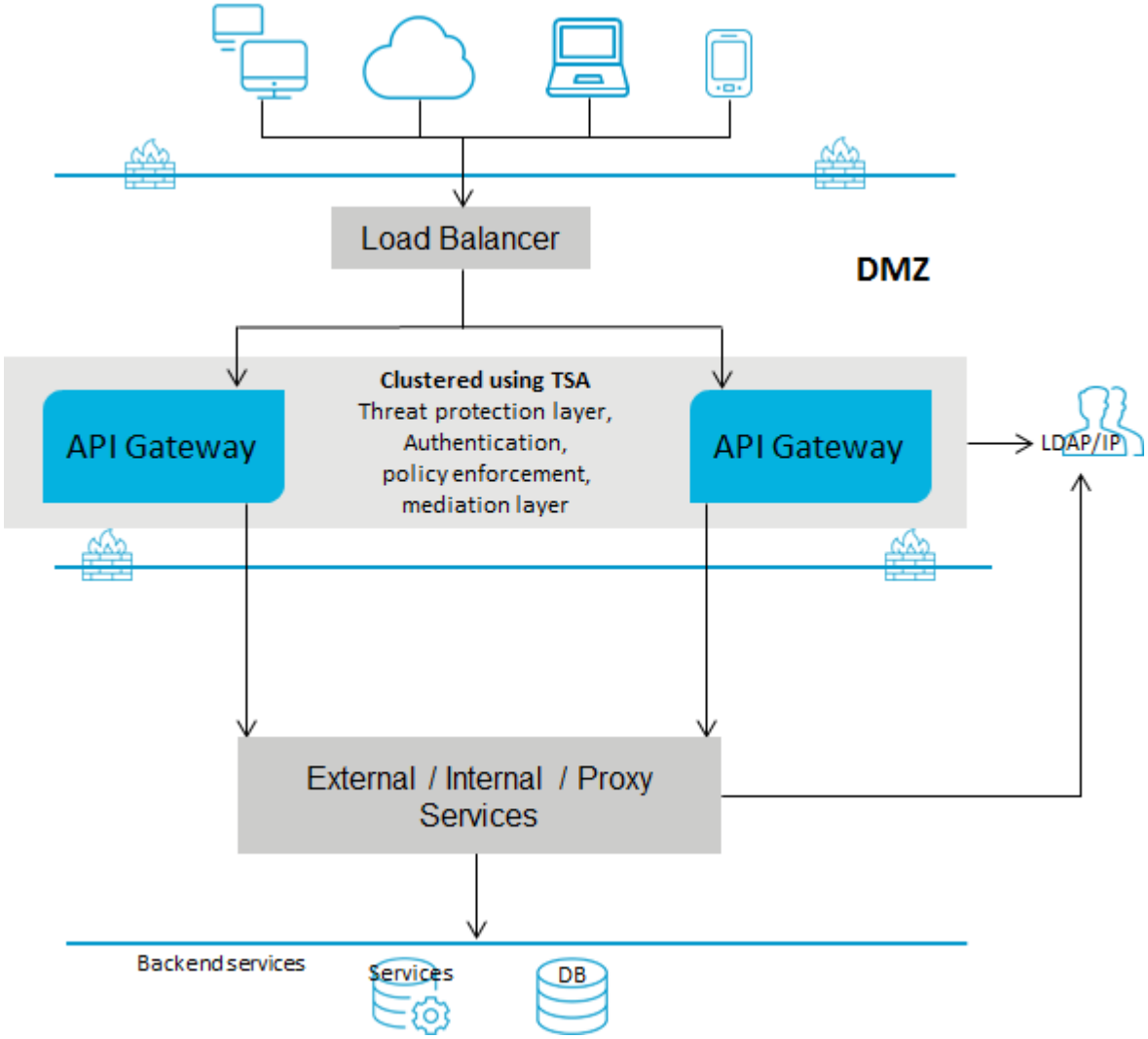
Deployment scenario 4: Single gateway for non-webMethods customers

This setup consists of:

- One or more advanced edition API Gateways clustered and connected to a load balancer in DMZ. A single API Gateway is used for enforcing all policies or rules. You can have multiple instances of API Gateways connected through a load balancer and clustered using Terracotta Server Array.

- The native services reside in the green zone behind the firewall. As the native services are directly invoked, you must open the native service port to the gateway network.

The following diagram provides an architectural overview of the API Gateway deployment for non webMethods customers:



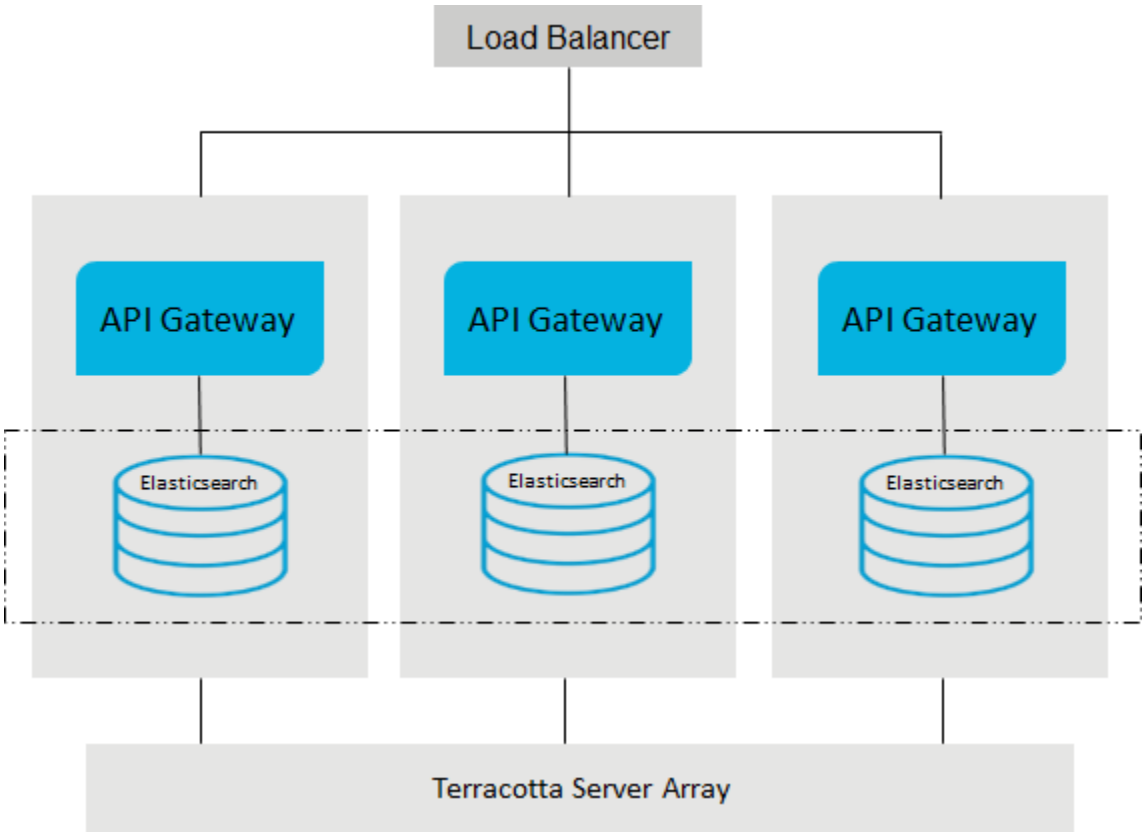
2 API Gateway Configuration

API Gateway Cluster Configuration

This section provides information about nodes and clusters in API Gateway and how to configure an API Gateway cluster after you have installed the product software. For installation procedures for the product software, see *Installing webMethods Products*.

Nodes and Clusters

API Gateway supports clustering to achieve horizontal scalability and reliability. The following figure illustrates an API Gateway cluster consisting of multiple API Gateway nodes.



Each API Gateway cluster node holds all the API Gateway components including UI, the API Gateway package running in webMethods Integration Server, and an Internal Data Store instance for storing assets. A load balancer distributes the incoming requests to the cluster nodes. The synchronization of the nodes is performed through a Terracotta server array and Internal Data Store clustering that is defined across the Internal Data Store instances.

As each node of an API Gateway cluster offers the same functionality, nodes can be added or removed from an existing cluster. The synchronization of any new node happens automatically. The synchronization includes configuration items, and runtime assets like APIs, policies, and applications. The synchronized runtime assets become active automatically.

Configuring an API Gateway Cluster

Configuring an API Gateway cluster requires the following:

- Configuring API Gateway cluster
- Configuring Internal Data Store cluster
- Configuring Terracotta Server array
- Configuring load balancer
- Configuring ports

API Gateway Configuration

You can enable API Gateway clustering through the API Gateway user interface. For more information on enabling API Gateway clustering, see *webMethods API Gateway User's Guide*

Internal Data Store Configuration

Each API Gateway cluster node consists of an Internal Data Store instance for storing run-time assets and configuration items. An Internal Data Store instance is a non-clustered Elasticsearch node. For a cluster configuration, the Internal Data Store instances should also be clustered using standard Elasticsearch clustering properties, by modifying the *SAG_root*/InternalDataStore/config/elasticsearch.yml file on each instance. For more information, see https://www.elastic.co/guide/en/elasticsearch/guide/current/important-configuration-changes.html and https://www.elastic.co/guide/en/elasticsearch/reference/2.3/index.html. The cluster name has to be specified and the cluster nodes have to be configured.

A sample configuration looks like follows:

```
cluster.name:"SAG_EventDataStore"
network.host:0.0.0.0
http.port:9240
transport.tcp.port:9340
node.master:true
discovery.zen.ping.unicast.hosts: ["apigateway1:9340","apigateway2:9340",
                                   "apigateway3:9340"]
```

Cluster Health

The health of the Internal Data Store cluster can be checked using the following URL:

http://daefermion4:9240/_cluster/health?pretty=true

Cluster health response example:

```
{
  "cluster_name" : "SAG_EventDataStore",
  "status" : "green",
  "timed_out" : false,
  "number_of_nodes" : 3,
```

```
  "number_of_data_nodes" : 3,
  "active_primary_shards" : 11,
  "active_shards" : 22,
  "relocating_shards" : 0,
  "initializing_shards" : 0,
  "unassigned_shards" : 0,
  "delayed_unassigned_shards" : 0,
  "number_of_pending_tasks" : 0,
  "number_of_in_flight_fetch" : 0,
  "task_max_waiting_in_queue_millis" : 0,
  "active_shards_percent_as_number" : 100.0
}
```

The response shows the status of the cluster and the number of its nodes. The following is an example sample response showing an unhealthy cluster status:

```
{
  "cluster_name" : "SAG_EventDataStore",
  "status" : "yellow",
  "timed_out" : false,
  "number_of_nodes" : 2,
  "number_of_data_nodes" : 2,
  "active_primary_shards" : 11,
  "active_shards" : 15,
  "relocating_shards" : 0,
  "initializing_shards" : 0,
  "unassigned_shards" : 7,
  "delayed_unassigned_shards" : 7,
  "number_of_pending_tasks" : 0,
  "number_of_in_flight_fetch" : 0,
  "task_max_waiting_in_queue_millis" : 0,
  "active_shards_percent_as_number" : 68.18181818181817
}
```

Here the Internal Data Store cluster state is yellow in the system and the number of nodes indicate that a cluster node is missing. An unhealthy cluster state can be caused by communication problems between the cluster nodes. To recover from an unhealthy state, Integration Server running the API Gateway with the missing node has to be restarted. A restart forces the Internal Data Store instance to rejoin the cluster.

For details on cluster health, see https://www.elastic.co/guide/en/elasticsearch/guide/current/_cluster_health.html.

Terracotta Server Array Configuration

API Gateway requires a Terracotta Server array installation. For more information see *webMethods Integration Server Clustering Guide* and the Terracotta documentation located at `http://www.terracotta.org/`

Load Balancer Configuration

A custom load balancer can be used for an API Gateway cluster. Here we use the load balancer nginx.

On a Linux machine, the load balancer configuration file `/etc/nginx/nginx.conf` is as follows:

```
user  nginx;
worker_processes  1;
error_log  /var/log/nginx/error.log debug;
pid        /var/run/nginx.pid;

events {
    worker_connections  1024;
}

http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;

    sendfile        on;
    #tcp_nopush     on;
    keepalive_timeout  65;
    gzip  on;

    upstream apigateway {
        server daefermion4:5555;
        server daefermion4:5556;
        server daefermion4:5557;
    }

    server {
        listen 8000;
        location / {
            proxy_pass http://apigateway;
        }
    }
}
```

Use `sudo nginx -s reload` or `sudo nginx -s start` to reload or start nginx respectively. In a test environment, the command nginx-debug is used for greater debugging. The load needs to be exposed through the firewall that is protecting the host the firewall is running on.

Ports Configuration

By default, API Gateway does provide synchronization of the port configuration across API Gateway cluster nodes. If you do not want the ports to be synchronized across API Gateway cluster nodes, set the `portClusteringEnabled` parameter available under ***Username*** **> Administration > General > Extended settings** in API Gateway to `false`.

Note:
When this parameter is set to `true`, all the existing port configurations except the diagnostic port (9999) and the primary port (5555) are removed.

Synchronization of ports configuration does not cover temporary disconnects of a node, therefore, to get a node synchronized, you must restart it. Also, if you do not remove the port configuration,

the port can be re-synchronized by performing another update on the same configuration. Therefore, to activate the ports synchronization, do the following:

1. Set the `portClusteringEnabled` parameter to `true`.
2. Restart all the cluster nodes.

Accessing the API Gateway User Interface

You can access the API Gateway UI in the following ways:

- Navigate to http://*host*:*port* where port is the HTTP port of API Gateway configured during installation. For example, http://host:9072.
- Log on to Integration Server administration console and click the home page of *WmAPIGateway* package.
- Log on to Integration Server administration console and click **API Gateway**... under **Solutions** menu.

Secure Internal Data Store for API Gateway

Internal Date Store, by default, is not secured. Elasticsearch Security and Search Guard are the two popular options to secure Internal Data Store. API Gateway, by default, ships open source version of Search Guard.

The high level steps to be performed to secure the Internal Data Store are:

1. Secure Internal Data Store server
2. Prepare various clients
3. Verify API Gateway functions properly

For more information on Elasticsearch Security, see https://www.elastic.co/products/x-pack/security.

For more information on Search Guard, see https://floragunn.com/searchguard/

For any troubleshooting information for Search Guard see, the troubleshooting reference.

Note:
Whenever there is a change in network or firewall settings, Internal Data Store might not be able to connect. You must restart Integration Server to connect to Internal Data Store.

Securing Internal Data Store

1. Shutdown API Gateway.
2. Open SAG_root/`InternalDataStore/bin/enable_ssl.bat/sh` and comment the last line `/plugins/search-guard-2/tools/sgadmin.bat/sh` and save the changes.

3. Copy `sagconfig` from *SAG_root*/IntegrationServer/instances/*Instance_Name*/packages/WmAPIGateway/config/resources/elasticsearch to *SAG_root*/InternalDataStore.

4. Execute *SAG_root*/InternalDataStore/bin/enable_ssl.bat/sh.

5. Execute *SAG_root*/InternalDataStore/bin/shutdown.bat/sh to shutdown Internal Data Store.

6. Open *SAG_root*/InternalDataStore/config/elasticsearch.yml. Remove all properties that start with `searchguard`, and add the following properties.

```
searchguard.ssl.transport.keystore_type: JKS
searchguard.ssl.transport.keystore_filepath: ../sagconfig/node-0-keystore.jks
searchguard.ssl.transport.keystore_alias: cn=node-0
searchguard.ssl.transport.keystore_password: a362fbcce236eb098973
searchguard.ssl.transport.truststore_type: JKS
searchguard.ssl.transport.truststore_filepath: ../sagconfig/truststore.jks
searchguard.ssl.transport.truststore_alias: root-ca-chain
searchguard.ssl.transport.truststore_password: 2c0820e69e7dd5356576
searchguard.ssl.transport.enforce_hostname_verification: false
searchguard.ssl.transport.resolve_hostname: false
searchguard.ssl.transport.enable_openssl_if_available: true

searchguard.ssl.http.enabled: false
searchguard.ssl.http.keystore_type: JKS
searchguard.ssl.http.keystore_filepath: ../sagconfig/node-0-keystore.jks
searchguard.ssl.http.keystore_alias: cn=node-0
searchguard.ssl.http.keystore_password: a362fbcce236eb098973
searchguard.ssl.http.truststore_type: JKS
searchguard.ssl.http.truststore_filepath: ../sagconfig/truststore.jks
searchguard.ssl.http.truststore_alias: root-ca-chain
searchguard.ssl.http.truststore_password: 2c0820e69e7dd5356576
searchguard.ssl.http.clientauth_mode: OPTIONAL

searchguard.authcz.admin_dn:
  - "CN=sgadmin"
```

7. Save the changes made to the file elasticsearch.yml

8. Execute *SAG_root*/InternalDataStore/bin/startup.bat/sh to start Internal Data Store.

9. Go to *SAG_root*/InternalDataStore/plugins/search-guard-2/tools and execute sgadmin.bat -cd ..\..\..\sagconfig\ -ks ..\..\..\sagconfig\sgadmin-keystore.jks -kspass 49fc2492ebbcfa7cfc5e -ts ..\..\..\sagconfig\truststore.jks -tspass 2c0820e69e7dd5356576 -nhnv -p 9340 -cn SAG_InternalDataStore.

   -p is the TCP port and -cn is the cluster name. Use / for shell scripts.)

10. Execute *SAG_root*/InternalDataStore/bin/shutdown.bat/sh. This is required only if the API Gateway is configured to start the Internal Data Store on startup which is the default configuration.

Now all TCP connections are secured with two-way authentication and HTTPS is enabled with basic authentication for the credentials Administrator and manage (with no two-way authentication) with the out of the box self-signed certificates.

Preparing the Clients

1. Preparing Kibana.

   a. Open *SAG_root*/profiles/IS_*Instance_Name*/apigateway/dashboard/config/kibana.yml and remove the comment tag and update the following properties:

      - elasticsearch.username
      - elasticsearch.password
      - elasticsearch.ssl.verificationMode
      - elasticsearch.ssl.certificateAuthorities : *file path of your root-ca.pem certificate*
      - elasticsearch.url: https://*hostname:port*

   b. Open uiconfiguration.properties file at <SAG_Home>\profiles\IS_default\apigateway\config and set apigw.kibana.autostart to `false`.

2. Preparing JVM.

   a. Import the *SAG_root*/ into the truststore configured or default store (*SAG_root*/jvm/jvm/jre/lib/security/cacerts) of JVM.

      This is required only for self-signed certificates.

3. Preparing Browsers.

   a. Import the *SAG_root*/ in the browser or accept the exception for self-signed certificates that is displayed when you access the browser for the first time.

Verifying API Gateway and Browsers

1. Verify API Gateway.

   a. Start API Gateway.

   b. Watch for exceptions in logs.

   You should be able to login and create APIs. You should be able to access the analytics page without any prompt for user credentials.

2. Verify the Browser.

   a. Navigate to https://*host*:*port*, where the port refers to the Internal Data Store HTTP port.

      A prompt for user credentials appears.

   b. Provide the user credentials.

      The basic details about the Internal Data Store node appears.

Connecting to an External Elasticsearch

In addition to using Internal Data Store as default data store for API Gateway, you can also use Elasticsearch to store the data. API Gateway uses an HTTP client to connect to Elasticsearch, to avoid version compatibility issues when different external versions of Internal Data Store and Elasticsearch are used. You can also connect to a cluster of Internal Data Store nodes or Elasticsearch servers.

Note:
If you use an external Elasticsearch with same version as Internal Data Store, then you can use the Kibana or dashboard that is shipped with API Gateway, else they have to be configured separately. If you have configured elastic search externally, then you have to configure Kibana externally.

You can configure the HTTP client in one of the following ways:

- While installing API Gateway using the installer with default settings. For details see, *Installing webMethods Products*.
- Updating the properties in the config.properties file to fine-tune performance or to configure an external Elasticsearch.

  Note:
  The values in WmAPIGateway/config/resources/elasticsearch/config.properties override the values that are configured in gateway-es-store.xml during runtime and the values in gateway-es-store.xml are not changed. During the first start-up of API Gateway, default values from gateway-es-store.xml are automatically copied to config.properties. From the next start-up of API Gateway, values from config.properties are used. Once the host is specified in config.properties the value is not over-written from gateway-es-store.xml.

If you have already configured the Internal Data Store or Elasticsearch, which by default have the transport client configured, you can configure the HTTP client as follows:

1. Navigate to WmAPIGateway/config/resources/elasticsearch/config.properties

   The config.properties file contains all the properties and Elasticsearch configurations.

2. Configure the following properties:

Property and Description

pg.gateway.elasticsearch.autostart

This property specifies whether the Elasticsearch starts automatically. If an external Elasticsearch is configured it has to be manually started. This property needs to be set to false to avoid Internal Data Store starting automatically.

Default value: true

pg.gateway.elasticsearch.client.http.response.size

This property is not present in API Gateway 10.3 by default.

API Gateway displays an error when the response size is large. This error statement can be viewed in the server log located at `SAG_root\IntegrationServer\instances\default\logs\server.log`. To avoid this error you can add this property with a required value.

You can compute the value from the response error that displays the response size in bytes.

For example, if you see an error for the response size that says `entity content is too long [105144960] for the configured buffer limit`, you can calculate the byte size in MB as follows: 105144960/(1024*1024) = 100.27 MB. To avoid this error you have to set a value greater than 100.27, so you can set it as 150.

pg.gateway.elasticsearch.config.location

This property specifies the location of the config file if you want to read port details from some other Elasticsearch config file

pg.gateway.elasticsearch.start.maxwait

This property specifies the maximum time of wait for Elastic Search starts.

pg.gateway.elasticsearch.hosts

Mandatory

This property lists Elasticsearch hosts and ports. The values are comma separated.

Default value: localhost:9240

Note:
Once a host is added to this property, this is the value that is used to connect to Elasticsearch and the host configured in gateway-es-store.xml is not considered.

pg.gateway.elasticsearch.http.keepAlive

Mandatory

This property creates the persistent connection between client and server.

Default value: true

Property and Description
pg.gateway.elasticsearch.http.connectionTimeout *Mandatory* This property specifies the time, in milliseconds, after which the connection times out. Default value: 10000
pg.gateway.elasticsearch.http.socketTimeout *Mandatory* This property specifies the wait time, in milliseconds, for a reply once the connection to Elasticsearch is established after which it times out. Default value: 30000
pg.gateway.elasticsearch.http.maxRetryTimeout *Mandatory* This property specifies the wait time, in milliseconds, for retries after which it times out. Default value: 100000 It is advisable to set max retry time for a request to (number of nodes * socketTimeOut)+connectionTimeout
pg.gateway.elasticsearch.http.keepAlive.maxConnections *Mandatory* This property specifies the maximum number of persistent connections that can be established between an API Gateway and Elasticsearch cluster. Default value: 50
pg.gateway.elasticsearch.http.keepAlive.maxConnectionsPerRoute *Mandatory* This property specifies the maximum number of persistent connections that can be established per HTTP route to an Elasticsearch server. Default value: 15
pg.gateway.elasticsearch.http.username This property specifies the user name to connect to Elasticsearch using basic authentication.
pg.gateway.elasticsearch.http.password This property specifies the password to connect to Elasticsearch using basic authentication.

Property and Description
pg.gateway.elasticsearch.https.keystore.filepath This property specifies the Keystore file path for establishing HTTPS communication with Elasticsearch.
pg.gateway.elasticsearch.https.truststore.filepath This property specifies the truststore file path for establishing HTTPS communication with Elasticsearch.
pg.gateway.elasticsearch.https.keystore.password This property specifies the Keystore password for establishing HTTPS communication with Elasticsearch.
pg.gateway.elasticsearch.https.keystore.alias This property specifies the Keystore alias for establishing HTTPS communication with Elasticsearch.
pg.gateway.elasticsearch.https.truststore.password This property specifies the truststore password for establishing HTTPS communication with Elasticsearch.
pg.gateway.elasticsearch.https.enabled This property specifies whether you want to enable or disable the HTTPS communication with Elasticsearch. Default value: false If this property is set to false none of the above properties related to HTTPS are respected.
pg.gateway.elasticsearch.outbound.proxy.enabled This property specifies whether you want to enable or disable outbound proxy communication. Default value: true
pg.gateway.elasticsearch.outbound.proxy.alias This property specifies the outbound proxy alias name used to connect to Elasticsearch.
pg.gateway.elasticsearch.https.enforce.hostname.verification This property enforces the host name verification for SSL communication. Default value: false
pg.gateway.elasticsearch.sniff.enable *Mandatory*

Property and Description
This property enables sniffers to add the other nodes in an Elasticsearch cluster to the client so that the client can talk to all nodes. Default value: true
pg.gateway.elasticsearch.sniff.timeInterval *Mandatory* This property enables adding the newly added Elasticsearch cluster nodes to existing REST client in a specified time interval in milliseconds. Default value: 60000

3. Restart API Gateway for the HTTP client to take effect.

Note:
If hosts and ports are changed for Elasticsearch then you have to udpate the appropriate Elasticsearch configuration for Kibana separately and restart the Elastic search server as well as Kibana.

Connecting to an External Kibana

You have to perform the following configurations if you are connecting to an external Kibana:

- Ensure the kibana version is compatible with the Elasticsearch version as Kibana and Elasticsearch have a one-to-one mapping. For details on version compatibility, see Support Matrix
- To connect to an external kibana you have to merge, replace or add the file kibana.yml from the installed location to `C:\API Gateway instance\profiles\IS_default\apigateway\dashboard\config`

  Note:
  You have to specify the Elasticsearch host and port details in the external kibana config file.
- If you are using a kibana version different than the one shipped with APi Gateway you have to specify the version in `C:\API Gateway instance\IntegrationServer\instances\default\packages\WmAPIGateway\config\resources\kibana\config\5\config.json`

  Note:
  If you are using kibana version lesser than 5 you have to specify the version in `C:\API Gateway instance\IntegrationServer\instances\default\packages\WmAPIGateway\config\resources\kibana\config\config.json`
- Turn off Kibana auto start by setting the property **apigw.kibana.autostart** to `false` located in `C:\API Gateway instance\profiles\IS_default\apigateway\config\uiconfiguration.properties`

- Specify the kibana URL in the property **dashboardInstance** located in `C:\API Gateway instance\IntegrationServer\instances\default\packages\WmAPIGateway\config\resources\beans\gateway-core.xml`

  For example:

```
<entry key="dashboard">
              <map key-type="java.lang.String">
                  <entry key="dashboardInstance" value="http://kibanahost:9405"/>
              </map>
          </entry>
```

Configuring Multiple Instances of API Gateway in a Single Installation

The instance creation script can be used to create another instance of API Gateway in the same installation. While creating another instance you can choose your preferred HTTP and HTTPS port for the API Gateway web application using `web.http.port` and `web.https.port` respectively and the back-end REST service endpoint port using `primary.port option`.

To create a new instance, run the following command:

```
is_instance.bat create -Dprimary.port=5656 -Dinstance.name=APIGateway
-Dweb.http.port=7474 -Dweb.https.port=7575 -Dpackage.list=WmAPIGateway
```

3 Docker Configuration

Overview

Docker is an open-source technology that allows users to deploy applications to software containers. A Docker container is an instance of a Docker image, where the Docker image is the application, including the file system and runtime parameters.

You can create a Docker image from an installed and configured API Gateway instance and then run the Docker image inside a Docker container. To facilitate running API Gateway in a Docker container, API Gateway provides a script to use to build a Docker image and then load or push the resulting Docker image to a Docker registry hosted on-premise or in webMethods Integration Cloud.

Support for API Gateway with Docker 1.12.11 and later is available on Linux and UNIX systems for which Docker provides native support.

For details on Docker and container technology, see Docker documentation.

Recommendations for using Docker with API Gateway

If you opt to run API Gateway in a Docker container, Software AG recommends the following:

- Create a Docker image for an installed, fully configured on-premise API Gateway. Make sure the server configuration is complete before creating the image.
- Consider a Docker image of API Gateway to be immutable. Software AG does not recommend making configuration or content changes on an API Gateway running in Docker container. Instead, make any changes on the on-premise API Gateway, recreate the Docker image, load or push the Docker image to the Docker repository, and then start a Docker container for the image.

Docker security

Docker, by default, has introduced a number of security updates and features, which have made Docker easier to use in an enterprise. There are certain guidelines or best practices that apply to the following layers of the Docker technology stack, that an organization can look at:

- Docker image and registry configuration
- Docker container runtime configuration
- Host configuration

For detailed guidelines on security best practices, see the official Docker Security documentation at https://docs.docker.com/engine/security/security/.

Docker has also developed Docker Bench, a script that can test containers and their hosts' security configurations against a set of best practices provided by the Center for Internet Security. For details, see https://github.com/docker/docker-bench-security.

For details on how to establish a secure configuration baseline for the Docker Engine, see Center for Information Security (CIS) Docker Benchmark (Docker CE 17.06).

For information on the potential security concerns associated with the use of containers and recommendations for addressing these concerns, see NIST SP 800-190 publication (Application Container Security Guide)

Prerequisites for Building a Docker Image

Prior to building a Docker image for API Gateway, you must complete the following:

- Install Docker client on the machine on which you are going to install API Gateway and start Docker as a daemon. The Docker client should have connectivity to Docker server to create images.
- Install API Gateway, packages, and fixes on a Linux or UNIX system using the instructions in Installing Software AG Products, and then configure API Gateway and the hosted products

Building the Docker Image for an API Gateway Instance

The API Gateway docker image provides an API Gateway installation. Depending on the existing installation the image provides a standard API Gateway or an advanced API Gateway. When running the image the API Gateway is started. The API Gateway image is created on top of an Integration Server image.

To build a Docker image for an API Gateway instance

1. Create a docker file for the Integration Server (IS) instance by running the following command:

```
is_container.sh createDockerfile [optional arguments]
```

Argument	Description
`-Dinstance.name`	Optional. IS instance name to include in the image. Default: default
`-Dport.list`	Optional. Comma-separated list of the ports on the instance to expose in the image. Default: 5555,9999
`-Dpackage.list`	Optional. Comma-separated list of Wm packages on the instance to include in the image. Default: all (this includes all of the Wm packages and the Default package)
`-Dinclude.jdk`	Optional. Whether to include the Integration Server JDK (true) or JRE (false) in the image. Default: true
`-Dfile.name`	Optional. File name for the generated docker file.

Argument	Description
	Default: Dockerfile_IS

2. Build the IS docker image using the docker file Dockerfile_IS by running the following command:

```
is_container.sh build [optional arguments]
```

Argument	Description
`-Dfile.name`	Optional. File name of the docker file to use to build the Docker image. Default: Dockerfile_IS
`-Dimage.name`	Optional. Name for the generated docker image. Default: is:micro

3. Create a docker file for the API Gateway instance from the IS image is:micro by running the following command:

```
apigw_container.sh createDockerfile [optional arguments]
```

Argument	Description
`instance.name`	Optional. API Gateway instance to include in the image. Default: default
`port.list`	Comma-separated list of the ports on the instance to expose in the image. Default: 9072
`base.image`	Name of the base Integration Server image upon which this image should be built.. Default: is:micro
`file.name`	Optional. File name for the generated docker file. Default: Dockerfile_IS_APIGW

The docker file is created under the root Integration Server installation directory.

4. Build the API Gateway docker image using the core docker file Dockerfile_IS_APIGW by running the following command:

```
apigw_container.sh build [optional arguments]
```

Argument	Description
`instance.name`	Optional. API Gateway instance to include in the image. Default: default
`file.name`	File name of the docker file to use to build the docker image. Default: Dockerfile_IS_APIGW
`image.name`	Optional. Name for the generated docker image that contains the custom packages. Default: is:apigw

The image is stored on the docker host. To check the image run the command `$ docker images`

Example

A sample shell script for creating and an API Gateway looks as follows:

```
echo "is createDockerfile ========================================================="
./is_container.sh createDockerfile
status=$?

if [ $status -ne 0 ]
then
    echo "Failed! status: $status"
    exit $status
fi

echo "is build ===================================================================="
./is_container.sh build
status=$?

if [ $status -ne 0 ]
then
    echo "Failed! status: $status"
    exit $status
fi

echo "apigw createDockerfile ======================================================"
./apigw_container.sh createDockerfile
status=$?

if [ $status -ne 0 ]
then
    echo "Failed! status: $status"
    exit $status
fi

echo "apigw build ================================================================="
./apigw_container.sh build
status=$?

if [ $status -ne 0 ]
```

```
then
    echo "Failed! status: $status"
    exit $status
fi
```

After running the steps the created images can be listed using the docker images command: `docker images`

```
REPOSITORY          TAG       IMAGEID            CREATED             SIZE
is                  apigw     af29373fc98a        15 hours ago        1.3GB
is                  micro     06e7c0de4807        15 hours ago        1.1GB
centos              7         36540f359ca3        12 days ago         193MB
```

Note:
The is:micro and therefore also the is:apigw images are based on the centos:7 image, which is available from the official CentOS repository

Retrieving Port Information of the API Gateway Image

- To retrieve the port information of the API Gateway image (is:apigw), run the following command :

  ```
  docker inspect --format='{{range $p,
  $conf := .Config.ExposedPorts}}
  {{$p}} {{end}}' is:apigw
  ```

  A sample output looks as follows:

  ```
  5555/tcp  9072/tcp  9999/tcp
  ```

Running the API Gateway Container

- Start the API Gateway container using the docker run command:

  ```
  docker run -d -p 5555:5555 -p 9072:9072 -name apigw is:apigw
  ```

  The docker run is parameterized with the IS and the webApp port exposed by the docker container. If the customer has configured different ports for IS and UI, the call has to be adapted accordingly. The name of the container is set to apigw.

  The status of the docker container can be determined by running the docker ps command: `docker ps`

  A sample output looks as follows:

  ```
  CONTAINER ID  IMAGE      COMMAND                  CREATED        STATUS    ->

  5b95c9badd59  is:apigw   "/bin/sh -c 'cd /s..."  15 hours ago   Up 15 hours

  ->
  PORTS                                                  NAMES
  0.0.0.0:5555->5555/tcp, 0.0.0.0:9072->9072/tcp, 9999/tcp   apigw
  ```

Load Balancer Configuration with the Docker Host

A port mapping is specified when you run the docker container. For example, running the docker container with the following command maps the IS port to the port 5858 on the docker host.

```
docker run -d -p 5858:5555 -p 9073:9072 --name apigw is:apigw
```

The host running the docker container is different from the host information within the docker container due to which the gateway endpoints exposed by API Gateway are not set correctly. To set this right you have to set up a load balancer configuration with the docker host and the mapped ports.

For the above example the following load balancer URLs are required:

- **Load balancer URL (HTTP)**: http://dockerhost:5858
- **Load balancer URL (WS)**: ws://dockerhost:5858
- **Web application load balancer URL**: http://dockerhost:9073

Note:
If the API Gateway UI port is mapped to a different port on the docker host, the API Gateway solution link in the IS Administration UI does not work.

Stopping the API Gateway Container

- Stop the API Gateway container using the docker stop command:

  ```
  docker stop -t90 apigw
  ```

  The docker stop is parameterized with amount of seconds required for a graceful shutdown of the API Gateway and the API Gateway docker container name.

  Note:
  The Docker stop does not destroy the state of the API Gateway. On restarting the docker container all assets that have been created or configured will be available again.

Managing API Gateway Images

The management of API Gateway images is performed by is_container.sh

- saveImage: To save an API Gateway image to a file (creating a tar ball from an image)
- loadImage: To load an image to a Docker registry (loading an image into a Docker registry from tar ball)

Configuring an API Gateway Docker Container Cluster

You can combine API Gateway Docker containers to form a cluster.

To configure an API Gateway Docker container cluster

1. Configure loadbalancer on the docker host.

   The custom loadbalancer is installed on the docker host. For more details on setting up the loadbalancer, see "Configuring an API Gateway Cluster" on page 19.

2. Configure Terracotta Server Array

   API Gateway requires a Terracotta Server Array installation. For details, see *webMethods Integration Server Clustering Guide* and Terracotta documentation (https://www.terracotta.org/generated/4.3.4/pdf/bigmemory-max/BigMemory_Max_Installation_Guide). The Terracotta Server Array on its own can be deployed as a Docker container.

3. Create the basic API Gateway docker image.

   For details on creating the API Gateway docker image, see "Building the Docker Image for an API Gateway Instance " on page 33.

4. Create cluster API Gateway docker image and enhance it with the cluster configuration.

   a. Adapt Integration Server configuration (SoftwareAG\IntegrationServer\instances\default\config\server.cnf).

   b. Adapt the wrapper configuration (SoftwareAG\profiles\IS_default\configuration\custom_wrapper.conf).

   c. Adapt the Internal Data Store configuration (SoftwareAG\InternalDataStore\config\elasticsearch.yml).

      For more details about the above configurations, see "API Gateway Cluster Configuration" on page 18

   The resulting docker file is used to create a docker image is_apigw_cluster.

5. Run API Gateway docker cluster

   Each container represents a cluster node. Ensure to start all containers together so that Internal Data Store instances running in the Docker containers can form a cluster.

4 Configuration Properties

Configuration Types and Properties

This section describes the configuration types and parameters that you must configure for API Gateway.

The configuration types are broadly classified as web-app, API Gateway package-level, and Elasticsearch configurations.

Web-app Configuration Properties

These properties are not cluster-aware and, hence, you must manually copy them to all the nodes.

General properties

Location: *SAG_root*/profiles/IS_*IS_Instance_Name*/apigateway/config/uiconfiguration.properties

apigw.auth.priority

API Gateway supports both Form-based and SAML-based authentication. If both are enabled, this property decides the login page to be displayed, by default, when a user visits the login page http://host:port/apigatewayui. A user can go to a specific login page using:

- Form: `http://host:port/apigatewayui/login`
- SAML: `http://host:port/apigatewayui/saml/sso/login`

Available values: `Form`, `SAML`.

Default value is `Form`.

apigw.auth.form.enabled

This property enables or disables Form-based authentication. If both SAML and Form are disabled, the value Form is retained by default.

Available values: `true`, `false`.

Default value is `true`.

apigw.auth.form.redirect

If a protected resource is accessed and the Form-based authentication is enabled, user is redirected to this page.

Default value is `/login`.

apigw.is.base.url

Host where the IS package is hosted. *localhost* is replaced by the hostname that is resolved through localhost.

Note:

The port changes to the default port of the Integration Server instance irrespective of HTTP or HTTPS.

Default value is `http://localhost:port`. Here, *port* denotes the port that is configured at the time of installation.

apigw.user.lang.default

This property denotes the language to be used in the API Gateway UI.

Default value is `en (English)`.

apigw.is.timeout

This property denotes the user session timeout value in minutes.

Default value is `90`.

SAML SSO properties

API Gateway user interface supports the following SAML values:

- **Profile**: Web Browser SSO
- **Protocol**: SAML Auth Request
- **Binding**: HTTP Post

The value sent in the NameID is used as logged in user Id. In the absence of a NameID, the attribute value with http://schemas.xmlsoap.org/ws/2005/05/identity/claims/upn namespace is used for the user id.

The attribute value with `http://schemas.microsoft.com/ws/2008/06/identity/claims/role` or `http://schemas.xmlsoap.org/claims/Group` can be used to pass the roles. These roles can be one or more roles supported by API Gateway.

An alternative option is to send any roles and map those roles to one of the API Gateway roles. The mapping must be done in the saml_groups_mapping.xml file located at *SAG_root*`/IntegrationServer/instances/`*IS_Instance_Name*`/packages/WmAPIGateway/config/resources/`. The format is `<group source="`*Some Role*`" target="`*API Gateway Role*`" />`. Example `<group source="API Managers" target="API-Gateway-Providers" />`.

You must then configure the *SAG_root*`/IntegrationServer/instances/`*IS_Instance_Name*`/config/is_jaas.cnf` file by replacing `com.wm.app.b2b.server.auth.jaas.SamlOSGiLoginModule requisite;` with `com.softwareag.apigateway.auth.saml.APIGatewaySamlLoginModule requisite;` and save the changes.

Configure the following properties, located at *SAG_root*`/profiles/IS_`*IS_Instance_Name*`/apigateway/config/uiconfiguration.properties`, to enable the SAML based SSO.

apigw.auth.saml.enabled

This property enables or disables SAML-based authentication.

Available values: `true`, `false`.

Default value is `false`.

apigw.auth.saml.redirect

Denotes the location of keystore. The keystore with self-signed certificates are shipped at *`SAG_root`*`/profiles/IS_`*`IS_Instance_Name`*`/apigateway/config/keystore/saml_sso.jks`. The keystore has 3 certificates with the following alias:

- sign: the password is signapigw
- encrypt: the password is encryptapigw
- default: the password is defaultapigw

The password for keystore is apigwstore.

Default value is `None`.

apigw.auth.saml.keystore.type

Denotes the keystore type.

Available values: `JKS`, `PKCS12`.

Default value is `JKS`.

apigw.auth.saml.keystore.pwd

Denotes the keystore password. On starting the web-app, this password is moved to passman secure store located at *`SAG_root`*`/profiles/IS_`*`IS_Instance_Name`*`/apigateway/config/passman` and the handle is maintained as part of this property. Tampering with the handle results in exceptions.

Default value is `None`.

apigw.auth.saml.signkey.alias

This is the certificate alias used for signing the SAML authentication request.

Default value is `None`.

apigw.auth.saml.signkey.pwd

Denotes the password for the certificate alias. On starting the web-app, this password is moved to passman secure store and the handle is maintained as part of this property. Tampering with the handle results in exceptions.

Default value is `None`.

apigw.auth.saml.encrypkey.alias

Denotes the certificate alias to be used for encrypting the SAML authentication request.

Default value is `None`.

apigw.auth.saml.encrypkey.pwd

Password for certificate alias used for encrypting the SAML authentication request. On starting the web-app, this password is moved to passman secure store and the handle is maintained as part of this property. Tampering with the handle results in exceptions.

Default value is `None`.

apigw.auth.saml.defaultkey.alias

This alias is used for signing and encryption if sign and encryption related alias are missed.

Default value is `None`.

apigw.auth.saml.defaultkey.pwd

Denotes password for default key alias. On starting the web-app, this password is moved to passman secure store and the handle is maintained as part of this property. Tampering with the handle results in exceptions.

Default value is `None`.

apigw.auth.saml.authreq.signed

Denotes whether to send the signed SAML authentication request.

Available values: `true`, `false`.

Default value is `true`.

apigw.auth.saml.assertion.signed

Denotes whether we expect the signed assertion to be sent by the IDP.

Available values: `true`, `false`.

Default value is `true`.

apigw.auth.saml.sp.id

Denotes service provider identity which is sent as part of SAML authentication request.

Default value is Host name of localhost.

apigw.auth.saml.ldp.metadata.url

Denotes the file URL of IDP metadata. Consult your IDP documentation on how to generate one.

Default value is `None`.

apigw.auth.saml.sp.metadata.url

Denotes the file URL of Gateway metadata. You can get the content from http://*host*:*port*/apigatewayui/saml/sso/metadata

Default value is `None`.

Kibana

Location : *SAG_root*/profiles/IS_*IS_Instance_Name*/apigateway/config/uiconfiguration.properties

apigw.kibana.autostart

Decides whether kibana should be started as part of web-app.

Available values: `true`, `false`.

Default value is `true`.

apigw.kibana.url

Denotes the URL where Kibana is running. *localhost* is replaced by the hostname that is resolved through localhost. The port and other configurations of the kibana can be changed from *SAG_root*/profiles/IS_*IS_Instance_Name*/apigateway/kibana-4.5.1/config/kibana.yml

Default value is `http://localhost:9405`

apigw.es.url

Denotes the URL where Internal Data Store (HTTP) is running. *localhost* is replaced by the hostname that is resolved through localhost.

Default value is `http://localhost:port`

port denotes the Internal Data Store HTTP port configured during installation.

Note:
If the configured host resolves to the host name of the localhost, the port changes to the HTTP port configured in the *SAG_root*/InternalDataStore/config/elasticsearch.yml file.

API Gateway Package Configuration Properties

API Gateway uses Internal Data Store (Elasticsearch) as its data repository. API Gateway starts the Internal Data Store instance, if configured, using the default configuration shipped and located at *SAG_root*/InternalDataStore/config/elasticsearch.yml

Note:
To run Internal Data Store instances in a cluster, the `elasticsearch.yml` file must be updated on each instance. For additional details, see Elasticsearch documentation.

Location : *SAG_root*/IntegrationServer/instances/*IS_Instance_Name*/packages/WmAPIGateway/config/resources/elasticsearch/config.properties

pg.gateway.elasticsearch.autostart

Denotes the flag to manage (start or stop) Internal Data Store as part of API Gateway. Set it to false if the start or stop of Internal Data Store is managed from outside the API Gateway.

Available values: `true`, `false`.

Default value is `true`.

pg.gateway.elasticsearch.start.maxwait

Denotes maximum time in seconds API Gateway waits for Internal Data Store to start and stop if autostart is set to `true`.

Default value is `300`.

pg.gateway.elasticsearch.config.location

Denotes the location of the config file. If you have to use a different config file, mention the location of the config file here.

Default value is *SAG_root*`/InternalDataStore/config/elasticsearch.yml`

Note:

- If the Internal Data Store hostname is same as localhost, then the system automatically modifies the value of `<prop key=cluster.name>` in *SAG_root*`/IntegrationServer/instances/`*IS_Instance_Name*`/packages/WmAPIGateway/config/resources/beans/gateway-datastore.xml` to `cluster.name` property in the elasticsearch.yml file.
- If the Internal Data Store hostname is same as localhost, then the system automatically modifies the port value of <value>localhost:9340</value> in *SAG_root*`/IntegrationServer/instances/`*IS_Instance_Name*`/packages/WmAPIGateway/config/resources/beans/gateway-datastore.xml` to `transport.tcp.port` property in the elasticsearch.yml file.
- Ensure that the cluster.name and transport.tcp.port properties are in synchronization if you encounter any errors.

Configuration Properties to Secure Elasticsearch

The section lists the configuration properties to secure Elasticsearch.

Server :*SAG_root*/InternalDataStore/config/elasticsearch.yml

Item	Description
TRANSPORT (2-Way authentication is enabled by default)	
`searchguard.ssl.transport.keystore_type`	Type of keystore Possible values: `JKS`, `PKCS12` Default value: `JKS`
`searchguard.ssl.transport.keystore_filepath`	Location where the keystore is stored

Item	Description
searchguard.ssl.transport. keystore_alias	Keystore entry name if there are more than one entries
searchguard.ssl.transport. keystore_password	Password to access keystore
searchguard.ssl.transport. truststore_type	Type of truststore Possible values: JKS, PKCS12 Default value: JKS
searchguard.ssl.transport. truststore_filepath	Location where the truststore is stored
searchguard.ssl.transport. truststore_alias	Truststore entry name if there are more than one entries
searchguard.ssl.transport. truststore_password	Password to access truststore
searchguard.ssl.transport. enforce_hostname_verification	If true, the hostname mentioned in the certificate is validated. Set this to false if it is general purpose self-signed certificate. Possible values: true, false Default value: true
searchguard.ssl.transport. resolve_hostname	Applicable only if above property is true. If true, the hostname is resolved against the DNS server. Set this to false if it is general purpose self-signed certificate. Possible values: true, false Default value: true
searchguard.ssl.transport.enable_ openssl_if_available	Use if OpenSSL is available instead of JDK SSL Possible values: true, false Default value: true
HTTP	
searchguard.ssl.http.enabled	Set this to true to enable the SSL for REST interface (HTTP)

Item	Description
	Possible values: `true`, `false` Default value: `true`
`searchguard.ssl.http.keystore_type`	Type of keystore Possible values: `JKS`, `PKCS12` Default value: `JKS`
`searchguard.ssl.http.keystore_filepath`	Location where the keystore is stored
`searchguard.ssl.http.keystore_alias`	Keystore entry name if there are more than one entries
`searchguard.ssl.http.keystore_password`	Password to access keystore
`searchguard.ssl.http.truststore_type`	Type of truststore Possible values: `JKS`, `PKCS12` Default value: `JKS`
`searchguard.ssl.http.truststore_filepath`	Location where the truststore is stored
`searchguard.ssl.http.truststore_alias`	Truststore entry name if there are more than one entries
`searchguard.ssl.http.truststore_password`	Password to access truststore
`searchguard.ssl.http.clientauth_mode`	Option to enable 2-way authentication. REQUIRE: Client requires the client certificate. OPTIONAL: Client may require the client certificate.. NONE: Ignores client certificate even if it is available. Possible values: `REQUIRE`, `OPTIONAL`, `NONE` Default value: `OPTIONAL`

Item	Description
Search Guard Admin	
`searchguard.authcz.admin_dn`	Search Guard maintains all the data in an index called **searchguard**. This is accessible only to users (client certificate is passed in sdadmin command) configured here.
Miscellaneous	
`searchguard.cert.oid`	All certificates used by the nodes on transport level should have the oid field set to a specific value. This oid value is checked by Search Guard to identify if an incoming request comes from a trusted node in the cluster. If `yes`, all actions are allowed. If `no`, privilege checks apply. Also, the oid is checked whenever a node wants to join the cluster. '1.2.3.4.5.5'

Server :*SAG_root*/InternalDataStore/sagconfig Folder

This folder contains all the self-signed certificates and default Search Guard security configurations. The default configuration allows **demouser** client certificate as valid user for TCP communication, and enforces basic authentication for the credentials Administrator and manage.

hash.bat/sh (*SAG_root*/InternalDataStore/plugins/search-guard-2/tools) tool shipped with Search Guard is used to hash the user passwords.

Client :*SAG_root*/IntegrationServer/instances/*Instance_Name*/packages/ WmAPIGateway/config/resources/beans/ gateway-datastore.xml

Item	Description
`searchguard.ssl.transport.enabled`	Indicates whether the client should use secure transport Possible values: `true`, `false` Default value: `true`

All TRANSPORT properties, which are mentioned above, are applicable for the client as well.

Client :*SAG_root*/profiles/IS_*Instance_Name*/apigateway/dashboard/config/ kibana.ymlprofiles/ IS_*Instance_Name*/apigateway/dashboard/config/kibana.yml

Item	Description
`elasticsearch.username`	Username to be used if basic authentication is enabled
`elasticsearch.ssl.verify`	Disable all SSL checks including the hostname and certificate validation. Set this to true if it is general purpose self signed certificates. Possible values: `true`, `false` Default value: `true`
`elasticsearch.ssl.cert`	Path of client certificate to be sent to Elastisearch. This is required if 2-way authentication is enabled.
`elasticsearch.ssl.ca`	If verify is true, this denotes the path to the CA certificate which is used to sign other certificates.
`elasticsearch.password`	Password to be used if basic authentication is enabled.

5 API Gateway Data Management

Data Backup and Restore

You can take regular backups of data stored in API Gateway Data Store to protect it against accidental data loss. You can take a backup of complete API Gateway data that includes analytics data and assets data or you can take a partial backup that includes the backup of assets data or backup of analytics data. When you take a backup, you copy the contents of the repository to a file or to a cloud storage. At a later stage, you can retrieve the contents of the backup and restore them to API Gateway.

Note:

- API Gateway supports incremental backup. For example, if you have taken a backup of 50 GB and there is an increase in backup to 52 GB, API Gateway takes a backup of the new 2 GB data added.
- While performing a backup, the database experiences additional load, therefore, Software AG recommends taking a backup when the usage is low so as to avoid performance degradation.
- While restoring the backup from the repository, API Gateway replaces the existing data in API Gateway.
- API Gateway is not accessible when database restore is in progress.

To take a complete or partial backup of the API Gateway data and restore it to API Gateway, you can use the API Gateway command line utility. To back up and restore the database in command line, use the `apigatewayUtil.bat` and the `apigatewayUtil.sh` files available in the *`Integration Server_directory`*`/instances/`*`instance_id`*`/packages/WmAPIGateway/cli/bin` folder for Windows and UNIX systems respectively.

API Gateway supports the following storage platforms:

- Network File System (NFS)
- Amazon Simple Storage Service (S3)

  Note:
  You must install the Amazon Web Services (AWS) cloud plugin if you want to use the Amazon S3 storage platform. To install the AWS cloud plugin, run the following command and restart Integration Server: *`Integration Server_directory`*`/InternalDataStore/bin/plugin install cloud-aws`

Using NFS Storage Platform

API Gateway uses NFS as the default repository in which the backup is stored. You can configure the repositories in which the backup is stored either in NFS or S3 cloud. However, you can create a single repository and place all the backup files in that repository.

- Taking a backup:

  By default, API Gateway stores the backup in the *`Integration Server_directory`*`/InternalDataStore/archives/` directory. For example, if you run the command, `apigatewayUtil.bat create backup -name` *`backup_file_name`*, to take a backup, the backup is saved in the *`Integration Server_directory`*`/InternalDataStore/archives/default` directory.

- Restoring a backup:

  To restore the data taken as backup to API Gateway, run the following command:
  `apigatewayUtil.bat restore backup -name backup_file_name`

  Note:
  Once the backup is restored, you must restart the API Gateway instance.

- Restoring backup to a new instance:

  1. Copy the data from *Integration Server_directory*/InternalDataStore/archives/ directory where the backup data is available.
  2. Go to the *Integration Server_directory*/InternalDataStore/archives/ directory where the backup data is to be restored and ensure that you delete any existing data in this directory.
  3. Paste the data in the *Integration Server_directory*/InternalDataStore/archives/ directory.
  4. Run the following command to restore the data: `apigatewayUtil.bat restore backup -name backup_file_name`

     Note:
     Once the backup is restored, you must restart the API Gateway instance.

- Specifying NFS Directory path:

  For API Gateways in a clustered environment, you must specify a NFS directory path. This directory path is a shared file location, which must be accessible to all the API Gateway nodes in the cluster, and not used by any other cluster.

  1. Before creating the NFS repository in Elasticsearch run the following command in all nodes of the cluster to configure the NFS directory path:

     ```
     apigatewayUtil.bat configure fs_path -path NFS_path
     ```

     For example,

     ```
     apigatewayUtil.bat configure fs_path -path c://sample//APIGATEWAY
     ```

  2. Restart Elasticsearch.
  3. Restart Integration Server to make the new NFS directory path available to store the backup, else the backup is stored in the default location.

- Specifying NFS Directory path if you are using external Elasticsearch

  1. Open the **elasticsearch.yml** file from the location, *Integration Server_directory*/InternalDataStore/config/.
  2. Provide the NFS directory location in the **path.repo** field.
  3. Save the changes.
  4. Restart Elasticsearch.

Using S3 Storage Platform

You can save your backups to S3 cloud.

- Creating a repository:

  1. To create a repository using S3, run the following command:

     `apigatewayUtil.bat configure manageRepo -file file_path`

     where *file_path* is the path where the S3 cloud details are specified.

     For example, `apigatewayUtil.bat configure manageRepo -file` Integration Server_directory/instances/instance_id/packages/WmAPIGateway/cli/bin/conf/gateway-s3-repo.cnf.

  2. Go to *Integration Server_directory*/instances/*instance_id*/packages/WmAPIGateway/cli/bin/conf.

  3. Open the `gateway-s3-repo.cnf` file. The following information appears:

     ```
     type=s3
     bucket=<s3-bucket-name>
     region=<s3-region>
     accesskey=<s3-access-key>
     secretkey=<s3-secret-key>
     basepath=<s3-base-path>
     ```

     Note:
     The supported regions are: US East (N. Virginia): `us-east-1`, US West (N. California): `us-west-1`, US West (Oregon): `us-west-2`, Asia Pacific (Seoul): `ap-northeast-2`, Asia Pacific (Singapore): `ap-southeast-1`, Asia Pacific (Sydney): `ap-southeast-2`, Asia Pacific (Tokyo): `ap-northeast-1`, EU (Frankfurt): `eu-central-1`, EU (Ireland): `eu-west-1`.

  4. Configure the S3 details in the `gateway-s3-repo.cnf` file, for example,

     ```
     bucket=apigateway-store
     region=us-west-1
     accesskey=123-test-123d-123
     secretkey=tests1232sk12312t
     ```

     After modifying the `gateway-s3-repo.cnf` file, run the following command:
     `apigatewayUtil.bat configure manageRepo -file <file_path>`

     For example, `apigatewayUtil.bat configure manageRepo -file` *Integration Server_directory*/instances/*instance_id*/packages/WmAPIGateway/cli/bin/conf/gateway-s3-repo.cnf

- Taking a backup:

  To take a backup of the data, run the following command: `apigatewayUtil.bat create backup -name` *backup_file_name*

  Note:
  The *backup_file_name* must be specified in lowercase.

- Restoring a backup:

To restore the data taken as backup to API Gateway, run the following command:
`apigatewayUtil.bat restore backup -name backup_file_name`

Note:
Once the backup is restored, you must restart the API Gateway instance.

- Restoring backup to a new instance:

1. Create a repository using S3 if not already created.

   Note:
   The S3 details which we provide in the `gateway-s3-repo.cnf` should point to the location where we have backup files which was taken earlier.

2. In case of multiple backups, run the following command to retrieve a list of backups:
   `apigatewayUtil.bat list backup`

3. Run the following command to restore the data using the required backup file:
   `apigatewayUtil.bat restore backup -name backup_file_name`

   Note:
   Once the backup is restored, you must restart the API Gateway instance.

Data Backup and Restore Commands

You can use a command-line interface (CLI) script to back up data that is stored on API Gateway Data Store. You can use the CLI script to restore database after a data failure or hardware failure on the API Gateway instance.

In a command line, go to `<Integration Server_directory>\instances\default\packages\WmAPIGateway\cli\bin` and run the following commands to take a database backup or restore the database from a backup:

If you want to...	Run the command
Backup data	`apigatewayUtil.bat create backup -name <backupFilename>`
Backup custom data	`apigatewayUtil.bat create backup -name <backupFile_name> -include <reference name>` Possible values for the parameter `reference name`: - `analytics` - to back up analytical data. - `assets` - to back up asset data.
Delete the backed up data	`apigatewayUtil.bat delete backup -name <backupFile_name>`
Restore the backed up data	`apigatewayUtil.bat restore backup -name <backupFile_name>`

If you want to...	Run the command
To retrieve all available backup files in the repository	`apigatewayUtil.bat list backup`
Delete a repository from API Gateway	`apigatewayUtil.bat delete manageRepo`
To retrieve all available repositories	`apigatewayUtil.bat list manageRepo`
To configure a repository in S3	`apigatewayUtil.bat configure manageRepo -file <file_path>`
Backup configurations and data	`apigw-backup-tenant.bat -backupDestinationDirectory <directory_path_to_store_backup_file> -backupFileName <backup_file_name_without_spaces> -backupTemplate <file_path_to_backup_template> -packagesTemplate <file_path_to_packages_template> -help`
Restore the backed up configurations and data	`apigw-restore-tenant.bat -backupFileName <backup_file_name_without_spaces> -backupDestinationDirectory <directory_path_to_store_backup_file> -filesToSkip <file_path_to_files_to_skip> -skipDataRestore -help`

Pre-requisites for Backing up and Restoring Data

The following points are to be considered in the API Gateway instances used for backup and restore:

- The Software AG root installation directory must be the same.
- The Integration Server instance name must be the same.
- The ports defined for the API Gateway webApp, Integration Server, and the Internal Data Store must be the same.

Backing up API Gateway Configuration Data

You can back up the API Gateway configuration information and data. At a later stage, you can restore the data that you backed up from API Gateway Data Store from the backup repository.

There are two types of backups: data backups and configuration backups.

The data backups are performed using the command tool `apigatewayUtil.[bat|sh] create backup`. The tool creates a backup archive of the Internal Data Store data.

The configuration backups are performed using the command tool `apigw-backup-tenant.[bat|sh]`. The tool creates a backup archive of the API Gateway configuration information and data. It is typically used in the disaster recovery scenarios to backup the data periodically and restore the data in the event of any disaster.

The default location of the backed up data is the *Integration Server_directory*/InternalDataStore/archives directory. You can write the backed up data to the InternalDataStore/archives folder mount from an external NFS or S3 service. API Gateway uses NFS as the default repository in which the backup is stored.

Pre-requisites for Backing up in a Distributed Environment

The following points are to be considered if API Gateway is installed in a clustered high availability setup:

- Configure a path to backup the Internal Data Store.
- Restart the Internal Data Store.

To backup configurations and data

- Run the command apigw-backup-tenant.[bat|sh]

  The syntax is of the format:
  C:/SoftwareAG/IntegrationServer/instances/default/packages/WmAPIGateway/cli/bin>apigw-backup-tenant.bat|sh -backupDestinationDirectory *directory_path_to_store_backup_file* -backupFileName *backup_file_name_without_spaces* -backupTemplate *file_path_to_backup_template* -packagesTemplate *file_path_to_packages_template* -help

  The input parameters are:

Parameter	Description
-backupDestinationDirectory	*Mandatory.* Path to the destination folder where you want to create the backup. Example: C:/SoftwareAG/AnotherBackupLocation
-backupFileName	*Optional.* Name of the file for the data backup. **Note:** The file name must not contain any spaces. Default value is apigw_disaster_recovery_backup If a file name is not specified, the default value is automatically set for this parameter.
-backupTemplate	*Optional.* Path to the configuration backup file. Example: C:/SoftwareAG/tenant-backup-template.txt **Note:**

Parameter	Description
	The paths specified in the backup file should be relative to the Software AG root installation folder `<SAGInstallDir>` Default value is: `C:/SoftwareAG/IntegrationServer/instances/instance_name/packages/WmAPIGateway/cli/bin/conf/tenant-backup-template.txt` The backup file (`tenant-backup-template.txt`) contains a list of configuration files and folders that need to be backed up, with one file or folder name in each line. The backup file can also include custom configuration files, that are defined specifically for a particular API Gateway instance.
`-packagesTemplate`	*Optional.* Path to the backup file. This file contains a list of custom packages to be backed up and includes one package name in each line. Default value is `conf/tenant-backup-packages.txt`
`-help`	*Optional.* Prints the help text summarizing the input parameters of this command.

The `apigw-backup-tenant` command creates the following entries in `-backupDestinationDirectory`:

- A ZIP file with the name specified for the parameter `-backupFileName`. This ZIP file contains the backup of API Gateway configurations.

  If a file name is not specified in this parameter, then the command creates a ZIP file named `-backupFileName.zip`.

- Directory named `default`. This directory contains the backup of Internal Data Store.

The command also creates a backup log file named `backup-tenant.log` in the *SAGInstallDir*`/IntegrationServer/instances/`*instance_name*`/packages/WmAPIGateway/cli/bin` directory.

Restoring API Gateway Configuration Data

You use the command tool `apigw-restore-tenant.[bat|sh]` to restore previously archived configuration files and data on an API Gateway instance.

Note:
Restoring overwrites the existing content in your API Gateway instance.

Pre-requisites for Restoring in a Distributed Environment

The following points are to be considered if API Gateway is installed in a clustered high availability setup:

- The Internal Data Store must be active in only a single node in the cluster.
- The API Gateway instance should be up and running.

To restore configurations and data

- Run the command `apigw-restore-tenant.[bat|sh]`

  The syntax is of the format:

  ```
  C:\SoftwareAG\IntegrationServer\instances\default\packages\WmAPIGateway\cli\bin>apigw-restore-tenant.bat|sh
  -backupFileName <backup_file_name_without_spaces> -backupDestinationDirectory
  <directory_path_to_store_backup_file> -filesToSkip <file_path_to_files_to_skip>
  -skipDataRestore -help
  ```

  The input parameters are:

Parameter	Description
`-backupDestinationDirectory`	*Mandatory.* Path to the destination folder where the backup is available. Example: `C:\SoftwareAG\AnotherBackupLocation` Specify this option only if you want the backup is available in a local directory or NFS itself.
`-backupFileName`	*Mandatory.* Name of the file for the data backup. **Note:** The file name must not contain any spaces.

`-filesToSkip`	*Optional.* Path to the data restore file.
	Example:
	`C:\SoftwareAG\skip-files.txt`
	Note:
	The paths specified in the restore file should be relative to the Software AG root installation folder `<SAGInstallDir>`
	Default value is:
	`C:\SoftwareAG\IntegrationServer\instances\`*`instance_name`*`\packages\WmAPIGateway\cli\bin\conf\skip-files.txt`

Parameter	Description
	The restore file (`skip-files.txt`) contains a list of configuration files and folders that need to be restored in the API Gateway instance, with one file or folder name in each line. If you do not want to restore a specific configuration file, you can remove it from this restore file.
`-skipDataRestore`	*Optional.* Skips restoring of the Internal Data Store (Elastic Search) data.
`-help`	*Optional.* Prints the help text summarizing the input parameters of this command.

The `apigw-restore-tenant` command creates a restore log file named `restore-tenant.log` in the `<SAGInstallDir>\IntegrationServer\instances\{instance_name}\packages\WmAPIGateway\cli\bin` directory.

Note:
The `apigw-restore-tenant` command automatically restarts the API Gateway instance.

Post-requisites in a Distributed Environment

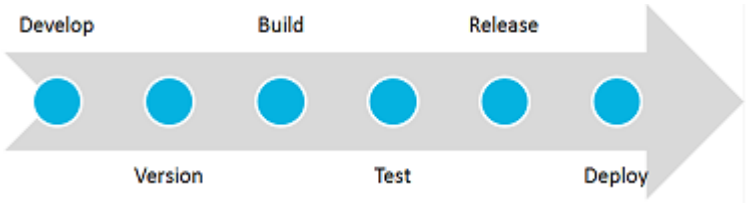
- If the Internal Data Store in active node does not start automatically, manually restart the Internal Data Store.
- Start the Internal Data Store in all other nodes in the cluster. This is important to synchronize the data on a restored API Gateway instance with all other nodes in the cluster.
- The API Gateway configuration files should be restored separately for each individual API Gateway instance in the clustered environment.

  Run the command `apigw-restore-tenant` with the parameter `-skipDataRestore`. This restores the API Gateway configuration files without restoring the Internal Data Store data in all other nodes in the cluster.

6 API Gateway Staging and Promotion

Introduction

API Gateway supports staging and promotion of assets. In a typical enterprise-level, solutions are separated according to the different stages of Software Development Lifecycle (SDLC) such as development, quality assurance (QA), and production stages. As each organization builds APIs for easy consumption and monetization, continuous integration (CI) and continuous delivery (CD) is an integral part of the solution. Where, CI is a development practice that requires developers to integrate code into a shared repository several times a day and CD is a software engineering approach in which teams produce software in short cycles, ensuring that the software can be reliably released at any time. Development of assets starts at the development stage and once the assets are developed, they are promoted to the QA stage for testing, after testing of the assets is complete, the assets are promoted to the deployment stage.



API Gateway provides tools and features to automate your CI and CD practices. Modifications made to the APIs, policies, and other assets can be efficiently delivered to the application developers with speed and agility.

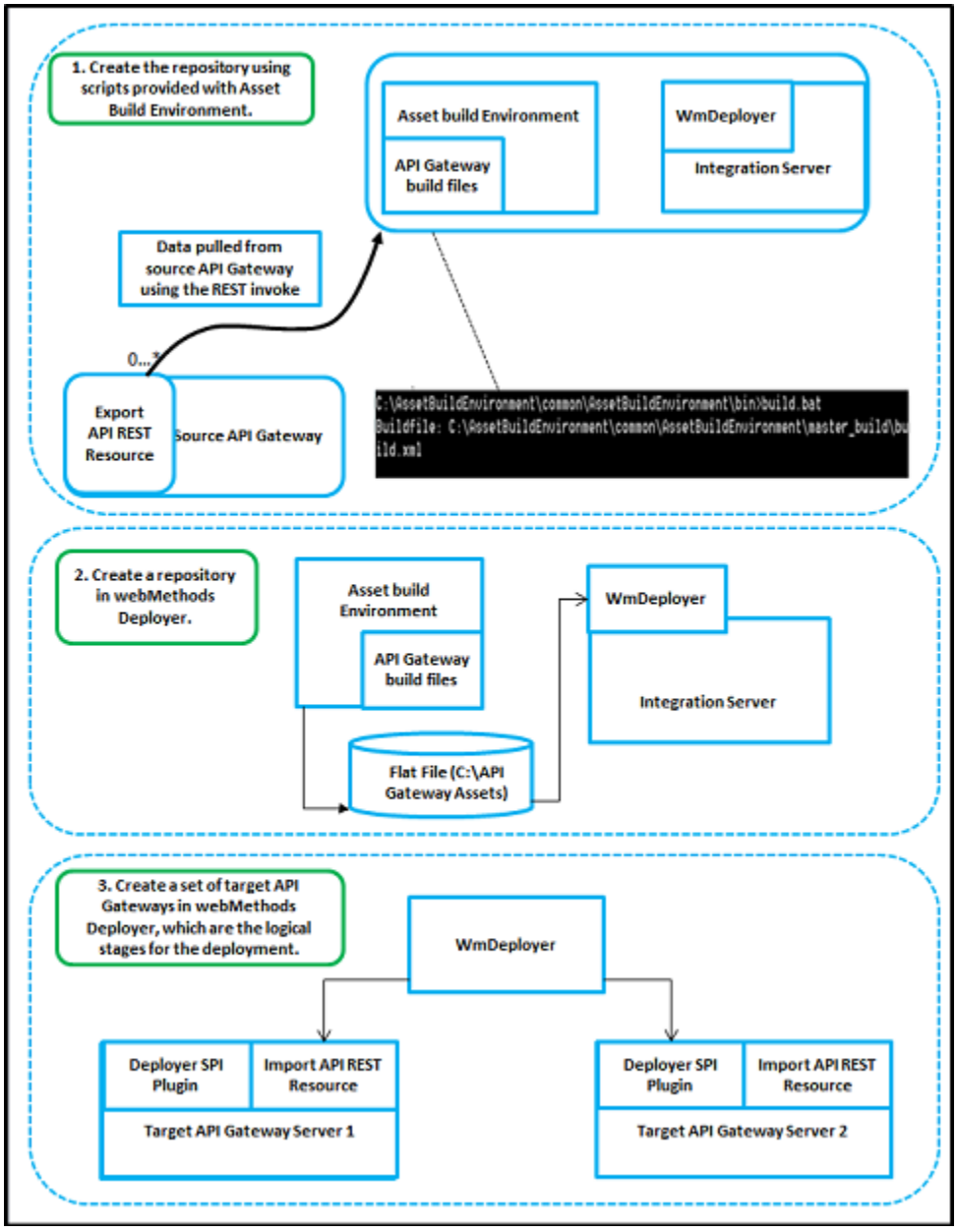
Note:
Software AG recommends you to have API Gateway instances across stages to be completely independent. For example, the API Gateway instances from the development stage and the API Gateway instances from the QA stage must not share any resources in common such as databases.

Asset Promotion in API Gateway

Promotion refers to moving API Gateway assets from one stage to another. You can promote assets using webMethods Deployer and Asset Build Environment.

You can promote API Gateway assets from one stage to the other using webMethods Deployer. webMethods Deployer is a tool you use to deploy user-created assets that reside on source webMethods runtimes or repositories to target webMethods runtime components (runtimes). For example, you might want to deploy assets you have developed on servers in a development environment (the source) to servers in a test or production environment (the target).

The high level steps involved are as follows:



For more information on promoting assets using webMethods Deployer , see *webMethods Deployer User's Guide*.

API Gateway staging and promotion allows you to:

- promote all the run time assets such as API Gateway APIs, aliases, applications, policies, or admin configurations across different stages.
- select and promote a subset of assets from one stage to another stage. For example, you can promote a single API and its policy dependencies from one stage to another.
- select dependencies involved while promoting an asset. For example, while selecting a service for promotion, you must also select the dependent policies, applications, and so on.
- modify values of attributes of selected aliases during promotion.

- roll back assets in case of failures.

Note:
During the promotion process ensure that both the source and the target system have the same master password.

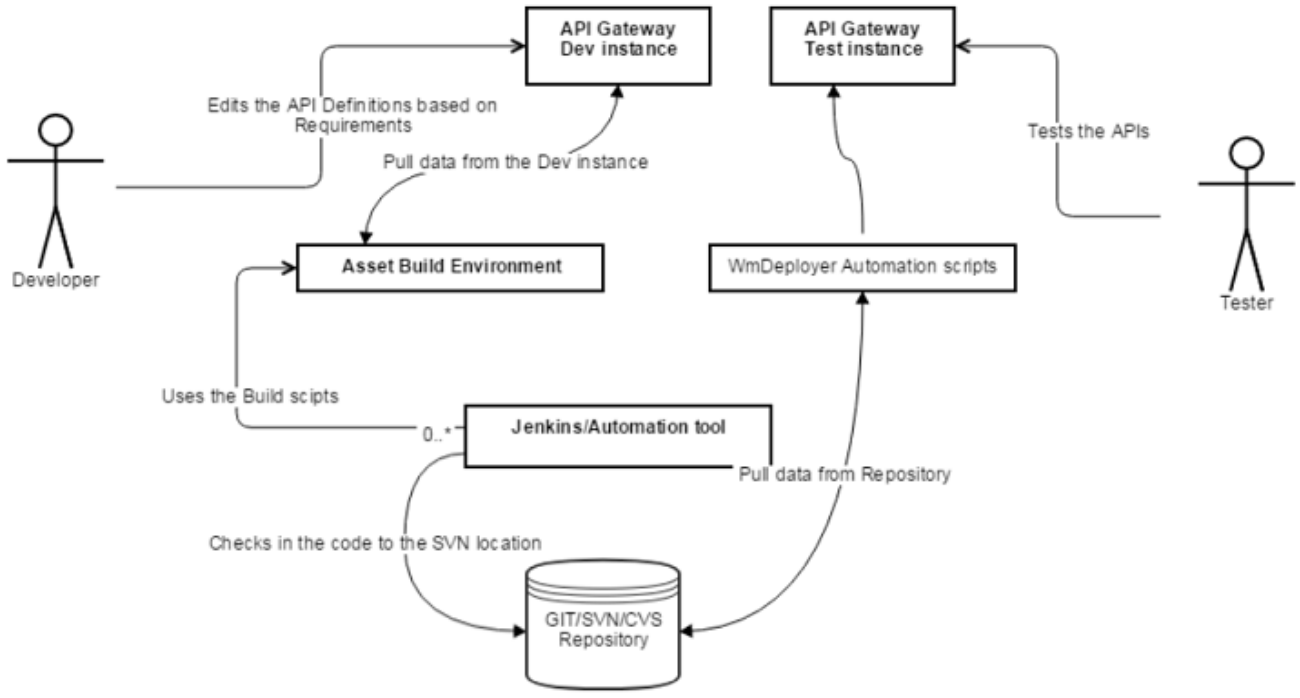
DevOps Usecase in API Gateway

API Gateway enables continuous integration (CI) and continuous delivery (CD) practices to be used for development, deployment, and promotion of the APIs, applications, other related assets, and for supporting the use of DevOps tooling.

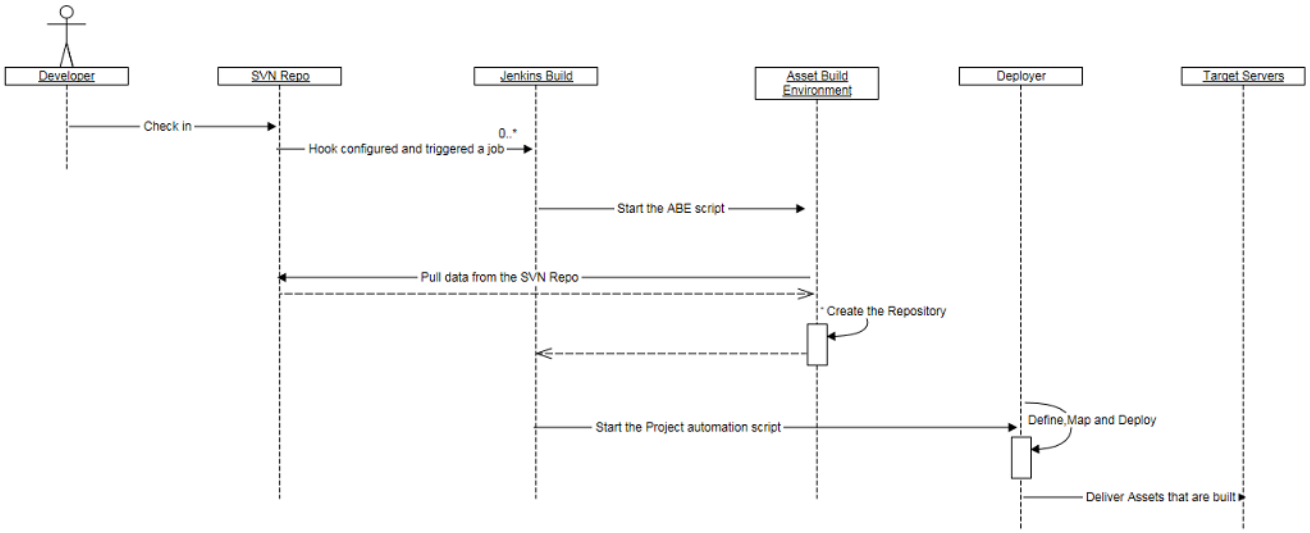
The API Gateway specific scripts that are provided as part of the Asset Build Environment and webMethods Deployer can be used by continuous integration tools like Jenkins. The sample flow is as follows:

1. The developer makes changes to a development API Gateway instance.
2. A Jenkins job then uses the build script to pull data from this development instance and push it to a version control system such as GIT.
3. Another job is used to pull it from a version control system and then use the webMethods Deployer scripts to directly push it to the test instance. In this way, the test instance always have the APIs.

Sample: Staging workflow



Sample: Staging call flow



For detailed information on promoting assets using webMethods Deployer , see *webMethods Deployer User's Guide*.

7 Mediator Migration to API Gateway

Migrating Mediator to API Gateway

API Gateway supports the migration of Mediator 9.7 and later; the earlier versions of Mediator should be migrated first.

Migrating Mediator Deployments to API Gateway

Existing Mediator deployments can be migrated to API Gateway by publishing the virtual service, applications, and runtime aliases to API Gateway. This lets you build an API Gateway runtime enforcement landscape in parallel to the existing Mediator landscape.

To migrate the existing Mediator deployments, perform the following procedure:

- For all installed Mediators:
  1. Stop Mediator.
  2. Install corresponding API Gateway.
  3. Migrate Mediator configuration to API Gateway.
- For all Mediator targets configured in CentraSite:
  1. Configure a corresponding API Gateway in CentraSite.
  2. Deploy all virtual services from the Mediator target to the corresponding API Gateway.
  3. (Optional) Undeploy all virtual services from the Mediator target.

Note:
The procedure assumes that the Mediators and the corresponding API Gateway provide the same endpoints. Therefore either the Mediator or its corresponding API Gateway can be up and running. If the endpoint compatibility is not required it is not necessary to stop the Mediators. Also the undeploying the Mediator deployments is optional. This means Mediator and API Gateway can be driven by CentraSite in parallel.

Migrating Mediator Configurations to API Gateway

To migrate existing Mediator configurations to API Gateway, perform the following procedure:

1. Run IS migration using the IS migration tool.

   For details of the IS migration tool, see *Upgrading Software AG Products*.

2. Run Mediator migration using the API Gateway migration tool.

   The API Gateway migration tool is available within the IS instance running the API Gateway. If API Gateway is running in the default IS instance the tool is available in the folder: `Install-Dir\IntegrationServer\instances\default\packages\WmAPIGateway\bin\migrate.`

   The script migrateFromMediator[.sh|.bat] has two parameters:

- Full path to Integration service installation running the Mediator to be migrated. (for example, E:\SoftwareAG\IntegrationServer)
- Name of the instance that is running the Mediator. (for example, default)

On Unix the script can be invoked as follows:

```
./migrateFromMediator.sh /opt/softwareag/IntegrationServer default
```

On Windows the script can be invoked as follows:

```
migrateFromMediator.bat C:\SoftwareAG\IntegrationServer default
```

3. Start API Gateway.

The Mediator configuration migration covers the following configuration items:

- Elastic Search
- SNMP
- Email
- HTTP Configuration
- Keystore Configuration
- Ports Configuration
- Service Fault
- Extended Settings

The following configuration items are not automatically migrated. The configuration of these items have to be done manually in API Gateway.

- Security Token Service (STS) Configuration
- `apig_rest_service_redirect` parameter: When you set this to `true`, the apig_rest_service_redirect in the extended Administration setting in API Gateway REST requests against the /mediator directive will be redirected to the /gateway directive. This means that REST requests can be sent to /mediator and to /gateway.

Note:
The Mediator configuration migration can only be applied to a fresh API Gateway installation once.